Please type a plus sign (+) inside this box [+]

PTO/SB/05 (1/98)
Approved for use through 9/30/2000 OMB 0651-0032
Patent and Trademark Office; U. S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applicaitions unter 37 CFR 1.53 (b))

Attorney Docket No.	82100
First Inventor or Application Identifier	CHRISTIAN L. HOULBERG
Title	NON-VOLATILE MEMORY FOR USE WITH AN ENCRYPTION DEVICE
Express Mail Label No.	XXXXXXXXXXXXX

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent applicaiton contents

ADDRESS TO: Assistant Commissioner for Patents
Box Patent Application
Washington, D.C. 20231

1. [X] *Fee Transmittal Form (e.g., PTO/SB/17) *(Submit an orginal, and a duplicate for fee processing)*
2. [X] Specification (Total Pages) 46 *(preferred arrangement set forth below)*
   - -Descriptive title of the invention
   - -Cross References to Related Applications
   - -Statement Regarding Fed sponsored R & D
   - -Reference to Microfiche Appendix
   - -Background of the Invention
   - -Brief Summary of the Invention
   - -Brief Description of the Drawings *(if filed)*
   - -Detailed Description
   - -Claim(s)
   - Apbstract of the Disclosure
3. [X] Drawing(s) *(35 U.S.C. 113)* (Total Sheets) 8
4. Oath or Declaration (Total Pages) 54
   - a. [X] Newly executed (orginal or copy)
   - b. [ ] *(for continuation / divisional with Box 17 completed)* Copy from a prior application(37CFR 1.63(d)) ***(Note Box 5 below)***
     - [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior applicaiton, see 37 C.F.R. 1.63 (d) (2) and 1.33 (b)
5. [ ] Incorporation By Reference (useable if Box 4b is checked) The entire disclosrue of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered to be part of the disclosure of the accomplnying application and is hereby incorporated by reference therein.
6. [ ] Microfiche Computer Program (Appendix)
7. Nucleotide and / or Amino Acid Sequence Submission (if applicable, all nccessary)
   - a. [ ] Computer Readable Copy
   - b. [ ] Paper Copy (identical to computer copy)
   - c. [ ] Statement verifying identity of above copies

OIPE FEB 17 2000 JC35 PATENT & TRADEMARK OFFICE

ACCOMPANYING APPLICATION PARTS

8. [X] Assignment Papers(cover sheet & document(s))
9. [ ] 37 CFR 3.73 (b) Statement [ ] Power of Attorney
10. [ ] English Translation Document (if applicable)
11. [ ] Information Disclosure [ ] Copies of IDS Citations
12. [ ] Preliminary Amendment
13. [ ] Return Receipt Postcard (MPEP 503) *(Should be specifically itemized)*
14. [ ] *Small Entity Statement(s) [ ] Statment filed in prior application
15. [ ] Certified Copy of Priority Document(s) *(if foreign priority is claimed)*
16. [ ] Other: .....

* *A new statement is required to be entitled to pay small entity fees, except where one has been filed in a prior application and is being relied upon.*

17.. If a **CONTINUING APPLICATION** check appropriate bos, and supply the requisite information below and in a preliminary amendment:

[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No. _____ / _____

Prior application information: *Examiner* *Group/Art Unit*

18. CORRESPONDENCE ADDRESS

[ ] Customer Number or Bar Code Label [X] Correspondence address below

Name: Commander - NAVAIRWARCENWPNDIV
Code 772000E

Address: 521 9TH Street

City: Point Mugu | State: CA | Zip Code: 93042-5001

Country: USA | Telephone: 805-989-8266 | Fax: 805-989-1695

Name (Print/Type)	David S. Kalmbaugh	Registration No. (Attorney/Agent)	29234
Signature	David Kalmbaugh	Date	02/08/00

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OF COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231

Navy Case No. 82,100

NON-VOLATILE MEMORY FOR USE WITH AN ENCRYPTION DEVICE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to a non-volatile memory interface for use with an encryption device. More particularly, the present invention relates a Non-Volatile memory circuit connected to an encryption device for storing the crypto key and the key loader for the encryption device.

2. Description of the Prior Art

The encryption device used for encrypting data to be transmitted to a ground station via a missile's telemetry system requires a crypto key to be loaded in the encryption device to permit the encryption of the data. The standard key loaders used by the military for crypto key loading are the KOI-18 and the KYK-13. The KOI-18 is a paper type reader that serially outputs the crypto key data and clock as a series of electrical pulses. The KYK-13 is an electrical device that can store up to three crypto keys with their corresponding check word. The KYK-13 outputs data in a manner which is similar to the KOI-18.

The missile's telemetry system encryption device includes a Non-Volatile Memory circuit which receives the

crypto key and check word from the key loader. Upon receiving the crypto key and check word the Non-Volatile Memory circuit will load the encryption device with the crypto key and also display the status of a load. When power is removed from the encryption device, only the Non-Volatile Memory circuit will retain the key data including the crypto key. When power is re-applied to the encryption system, the Non-Volatile Memory circuit automatically reloads the encryption device with the key data. The crypto key will remain in the Non-Volatile Memory circuit until the the key is erased from the circuit.

While the Non-Volatile Memory circuit used in the past perform their intended function of key data storage adequately, these circuits generally require substantially more space than is currently available on today's state of the art missile encryption systems. There is now a need to significantly reduce the size of Non-Volatile Memory circuit used with a missile's telemetry system encryption device.

SUMMARY OF THE INVENTION

The present invention overcomes some of the difficulties of the prior art including those mentioned above in that it comprises a relatively simple in design yet

highly effective Non-Volatile Memory circuit for use with a missile's telemetry encryption system.

The present invention comprises a Non-Volatile Memory circuit which functions as an interface between a key loader and an encryption device. Included in the Non-Volatile Memory circuit is a Flash/EEPROM 8-bit Microcontroller which has an EEPROM suitable for storage of a crypto key and its corresponding checkword and also a backup crypto key and checkword. Connected to the microcontroller is a 4 MHz clock signal generator which supplies the master clock signal to the microcontroller. A pair of light emitting diodes are also connected to the micrcontroller to indicate the status of a load of the crypto key and checkword within the microcontroller as well as the status of an erase of the crypto key and checkword from the microcontroller. The microcontroller is also connected to the telemeter transmitter for the missile. This allows the micrcontroller to turn off the transmitter during a key load which prevents transmission of the crypto key and its corresponding checkword.

When the microcontroller completes a load of the crypto key from its internal EEPROM to the encryption device and upon launch of the missile, the software within the microcontroller erases the crypto key and its corresponding

checkword from its EEPROM. This prevents an enemy force from retrieving the crypto key and its corresponding checkword from the missile after launch. The microcontroller can also erase the crypto key and its corresponding checkword from its EEPROM upon receiving an active erase signal from the missile.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a missile's telemetry encryption system and external key loader;

FIG. 2 is a detailed electrical diagram of the Non-Volatile Memory circuit of FIG. 1 which comprises the present invention;

FIGS. 3A-3C illustrate timing and data waveforms associated with a data transfer between the key loader and the Non-Volatile Memory circuit of FIG. 1; and

FIGS. 4-9 depicts a flow chart for the software used by the 8-bit microcontroller of FIG. 2 to load a crypto key with its corresponding check word into the encryption device of FIG. 1.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to FIGS. 1 and 2, there is shown a missile's telemetry encryption system which includes a key loader 22

for loading a crypto key with its corresponding check word into a Non-Volatile Memory circuit 20. The key loader 22 may be either be a KOI-18 and a KYK-13 key loader. It should be noted that the KYK-13 key loader can store three crypto keys with their corresponding check words.

Non-Volatile Memory circuit 20 is connected to a KVG-68 encryption device 24 which allows Non-Volatile Memory circuit 20 to load a crypto key with its corresponding check word into the encryption device 24. The encryption device is connected to a telemeter transmitter 26 which transmits encrypted telemetry data from an encryption device 24 to a ground station.

As shown in FIG. 2, Non-Volatile Memory circuit 20 includes an 18-pin Flash/EEPROM 8-bit Microcontroller 32 which stores the crypto key and corresponding check word used by encryption device 24. The 18-pin Flash/EEPROM 8-bit microcontroller 32 used in the preferred embodiment of the present invention is a Model PIC16F84 commercially available from Microchip Technology Inc. of Phoenix, Arizona. Connected to microcontroller 32 is a 4 MHz clock signal generator 34 which supplies the master clock signal to microcontroller 32.

Referring to FIGS. 1, 2 and 4, a power up circuit comprising a pair of resistors R10 and R11, a diode D2 and a

capacitor C1. When power is first applied to microcontroller 32 upon powering up Non-Volatile Memory circuit 20 a logic zero is supplied to the /MCLR input of microcontroller 32 clearing microcontroller 32. This logic zero then transitions to a logic one which results in microcontroller 32 executing the main routine (FIG.4) of the computer software of Appendix A.

The main routine begins at program step 40, proceeding to program step 42 which is the initialize_system routine illustrated in FIG. 5 and also included in the nvmem.c module of the software of Appendix A. The initialize system routine sets all of the port output signals of microprocessor 32 to their initial condition (program step 60); initializes the interrupts for microprocessor 32 (program step 62) and initializes the test indicators LEDS 36 and 38 (program step 64). During program step 66 the EEPROM of microprocessor 32 is scanned to determined if a valid crypto key was previously loaded into the EEPROM of microprocessor 32. If a valid key is detected an internal flag is set which allows for a load of the key into encryption device 24 by the software of Appendix A.

During initialization the /VAR_REQ output from microprocessor 32 is set high since this signal is active low signal.

At this time it should be noted that the software of Appendix A is adapted for processing two KGV-68 although only one is illustrated in FIG. 1. In a security upgrade configuration the software operates in a manner which allows two KGV-68 encryption units to be loaded with a crypto key and its corresponding check word. It should be noted that while FIG. 1 only shows one KVG-68, the non-volatile memory comprising the present invention may be easily modified to accommodate to KVG-68 encryption units.

After initialization the ERASE output from microprocessor 32 is set high since this signal is an active low signal which turns off LED 38. After initialization the STATUS output from microprocessor 32 is also set high since this signal is an active low signal which turns off LED 36. During initialization of microcontroller 32 the ERASE output and STATUS output from microprocessor 32 are pulsed to test the operation of LEDS 36 and 38. Setting the ERASE output of microprocessor 32 high indicates that the crypto key has not been erased from microprocessor 32. Setting the STATUS output of microprocessor 32 high indicates that encryption device 24 is not loaded.

The XMTR_DISABLE output from microprocessor 32 is set high during initialization to disable transmitter 26. The ENCR_SENSE_IN output from microprocessor 32 is set low

during initialization indicating that the KVG-68 encryption device 24 is not being loaded. The ENCR_FCLK and ENCR_FDATA outputs from microprocessor 32 are set high during initialization. The clock signal provided by microcontroller 32 at the ENCR_FCLK output from micrcontroller 32 has an active falling edge necessitating that the signal be set high during initialization of micrcontroller 32. Setting the ENCR_FDATA output from microprocessor 32 high results in "0" at the ENCR_FDATA output of microprocessor 32.

Referring to FIGS. 1, 2, 4 and 6, during program step 44, the software of Appendix A test for the presence of key loader 22. The SENSE_IN line is monitored by microcontroller 32 to determine the presence of key loader 22. When the SENSE_IN line is high resulting in a "1" at the RA0 input of microcontroller 32, the software of Appendix A proceeds to the eeprom_key_load routine of FIG. 6.

During program step 70 transmitter 26 is disabled by micrcontroller 32 to prevent possible transmission of the crypto key. During program step 72 the /VAR_REQ output from microprocessor 32 is set low to request the checkword from key loader 22. During program step 74 the checkword is loaded into the EEPROM of microcontroller 32. Program step

78 waits for indication that the key will be transferred from key loader 22 to the EEPROM of microcontroller 32 with the key being loaded into the EEPROM of micrcontroller 32 during program step 82. Micrcontroller 32 and the software of Appendix A also duplicate the key and checkword in a backup location in the EEPROM of micrcontroller 32.

During program step 84 an indication is provided that the key is present by clearing the ERASE LED 36 turning off the ERASE LED 36. During program step 86, transmitter 26 is enabled by microcontroller 32. During program step 46, the software of Appendix A returns to the main program of FIG. 4.

During program step 48, the software of Appendix A checks for the presence of the key. If the key is not present, i.e. the key is not accurately read into microcontroller 32, the software returns to program step 44 to determine if the key loader 22 is present. When key loader 22 is present, the software of Appendix A will again load the key.

When the key is correctly loaded into micrcontroller 32, the software of Appendix A proceeds to program step 50 which is the KGV load attempt decision. When a decision is made to load encryption unit 24, the software of Appendix A proceeds to the routine kgv_key_load of FIG. 7 (program step

52). During program step 90, transmitter 26 is disabled. During program step 92 the KGV sense input (ENCR_SENSE_IN) is set active, i.e. the logic "one" state, to start a load of the crypto key with its corresponding check word. Encryption unit 24 then responses with an active low variable request signal (/ENCR_VAR_RQ) to microcontroller 32 (program step 94). During program step 96, there is a set up for the start of the key load interrupt within microcontroller 32. During program step 98 an internal timer within microcontroller 32 is initialized and the key load interrupt is enabled for the key loading process.

During program step 100 there is an indication within micrcontroller 32 that the key should be present. During program step 102 a wait routine occurs which allows for completion of the key load process. When the key load process is complete, which is an internal indication from the interrupt routine, the KGV sense input (ENCR_SENSE_IN) is set inactive, i.e. a logic "zero" state (program step 104).

During program step 106, the software of Appendix A increments the count to keep track of the key load attempts. During program step 108 the software of Appendix A sets a flag to use the backup key on the next attempt. A second crypto key with its corresponding check word are stored in

the EEPROM of microcomputer 32. This backup key is utilized in the event that the primary key is not functional.

During program step 110, the software of Appendix A determines whether the key is loaded by testing random compare input (/ENCR_RAN_CP) to microcomputer 32. The answer will be no since there is a requirement that the routine kgv_key_load of FIG. 7 be processed twice to load the crypto key and the checkword into encryption device 24.

At this time it should be noted that the checkword is loaded first followed by the crypto key. During program step 112 the software of Appendix A determines whether there has been more than three attempts to load the checkword and the crypto key, which equates to six loops of the routine kgv_key_load of FIG. 7. If the answer is "yes" then transmitter 26 is enabled during program step 114. When this occurs the light emitting diode 36 will blink (program step 116) to indicate that microcontroller 32 has been unsuccessful in its attempt to load encryption device 24.

When a load of encryption device 24 is successful light emitting diode 36 remains on (program step 116). Program step 118 the software of Appendix A sets an internal flag indicating that a key load has been attempted. This prevents an inadvertent return to the routine kgv_key_load of FIG. 7.

The software of Appendix A next returns to main routine of FIG. 4. During program step 54, a determination is made as to whether or not the key should be erased. When the ERASE input to micrcontroller 32 is high (RA4 input to microcontroller 32), the microcontroller 32 erases the checkword and the crypto key as well as its backup from the EEPROM within microcontroller 32. Five random writes are performed within the EEPROM within microcontroller 32. This logic one signal, i.e. ERASE signal is provided by the loader interface 28 or the missile interface 30 to the RA4 input of micrcontroller 32. The signal provided by the missile interface 30 is substantially higher than the digital logic levels necessitating the use of additional resistor R9 in the LAUNCH line connecting missile interface 30 to microcontroller 32.

Referring to FIG. 8, the routine for erasing the EEPROM within microcontroller 32 is erase_key. Program step 120 debounces the erase indication signal provided to the RA4 input to microcontroller 32. Whenever the signal provided to the RA4 input to microcontroller 32 is a logic "one", the software of Appendix A proceeds to program step 124 erasing the crypto key with its corresponding check word from the EEPROM within microcontroller 32. The erase light, i.e. light emitting diode 38 is set, and the load status is

displayed during program step 124.

From the foregoing, it may readily be seen that the present invention comprises a new, unique and exceedingly causeway mooring apparatus for use in non-volatile memory for use with an encryption device which constitutes a considerable improvement over the known prior art. Many modifications and variations of the present invention are possible in light of the above teachings. It is to be understood that within the scope of the appended claims the invention may be practiced otherwise than as specifically described.

What is claimed is:

1. An apparatus for providing a crypto key and an associated checkword of said crypto key to an encryption device for a telemeter system of a missile, said apparatus comprising:

loading means for generating said crypto key and said associated checkword;

control means connected to said loading means to receive said crypto key and said associated checkword from said loading means, said control means sending a first logic signal to said loading means to effect a transfer of said crypto key and said associated checkword from said loading means to said control means for storage within said control means;

said control means being connected to said encryption device, said control means sending a second logic signal to said encryption device to initiate a load of said crypto key and said associated checkword into said encryption device;

said control means receiving from said encryption device a third logic signal, said control means, responsive to said third logic signal, loading said crypto key and said associated checkword into

said encryption device;

said control means being connected to a transmitter for the telemeter system of said missile, said control means providing a fourth logic signal to said transmitter to disable said transmitter when said crypto key and said associated checkword are loaded into said encryption device preventing said crypto key and said associated checkword from being transmitted by said transmitter; and

said control means being connected to a missile interface within said missile to receive a fifth logic signal from said missile interface upon a launch of said missile, said control means, responsive to said fifth logic signal, erasing said crypto key and said associated checkword from said control means.

2. The apparatus of claim 1 wherein said control means comprises an 8-bit Microcontroller.

3. The apparatus of claim 1 wherein said control means includes an EEPROM for storing said crypto key and said associated checkword and a copy of said crypto key and said associated checkword.

4. The apparatus of claim 1 further comprising a light emitting diode connected to said control means, said light emitting diode displaying a status for a load of said crypto key and said associated checkword into said encryption device.

5. The apparatus of claim 1 further comprising a light emitting diode connected to said control means, said light emitting diode displaying a status for an erase of said crypto key and said associated checkword from said microcontroller.

6. An apparatus for providing a crypto key and an associated checkword of said crypto key to an encryption device for a telemeter system of a missile, said apparatus comprising:

a key loader having said crypto key and said associated checkword stored therein;

a microcontroller connected to said key loader to receive said crypto key and said associated checkword from said key loader, said microcontroller sending a first variable request signal to said key loader to effect a transfer of

said crypto key and said associated checkword from said key loader to said microcontroller for storage within said microcontroller;

said microcontroller being connected to said encryption device, said microcontroller sending a sense in signal to said encryption device to initiate a load of said crypto key and said associated checkword into said encryption device;

said microcontroller receiving from said encryption device a second variable request signal, said microcontroller, responsive to said second variable request, loading said crypto key and said associated checkword into said encryption device; and

said microcontroller being connected to a transmitter for the telemeter system of said missile, said microcontroller providing a transmitter disable signal to said transmitter to disable said transmitter when said crypto key and said associated checkword are loaded into said encryption device preventing said crypto key and said associated checkword from being transmitted by said transmitter.

7. The apparatus of claim 6 wherein said microcontroller comprises an 8-bit Microcontroller.

8. The apparatus of claim 6 wherein said microcontroller includes an EEPROM for storing said crypto key and said associated checkword and a copy of said crypto key and said associated checkword.

9. The apparatus of claim 6 further comprising a light emitting diode connected to said microcontroller, said light emitting diode displaying a status for a load of said crypto key and said associated checkword into said encryption device.

10. The apparatus of claim 6 wherein said microcontroller is connected to a missile interface within said missile to receive a launch signal from said missile interface upon a launch of said missile, said microcontroller, responsive to said launch signal, erasing said crypto key and said associated checkword from said microcontroller.

11. The apparatus of claim 10 further comprising a light emitting diode connected to said microcontroller, said

light emitting diode displaying a status for an erase of said crypto key and said associated checkword from said microcontroller.

12. The apparatus of claim 6 wherein said microcontroller is connected to a loader interface within said missile to receive an erase signal from said loader interface, said microcontroller, responsive to said erase signal, erasing said crypto key and said associated checkword from said microcontroller.

13. An apparatus for providing a crypto key and an associated checkword of said crypto key to an encryption device for a telemeter system of a missile, said apparatus comprising:

a key loader having said crypto key and said associated checkword stored therein;

a microcontroller connected to said key loader to receive said crypto key and said associated checkword from said key loader, said microcontroller sending a first variable request signal to said key loader to effect a transfer of said crypto key and said associated checkword from said key loader to said microcontroller for

storage within said microcontroller;

said microcontroller being connected to said encryption device, said microcontroller sending a sense in signal to said encryption device to initiate a load of said crypto key and said associated checkword into said encryption device;

said microcontroller receiving from said encryption device a second variable request signal, said microcontroller, responsive to said second variable request, loading said crypto key and said associated checkword into said encryption device;

said microcontroller being connected to a transmitter for the telemeter system of said missile, said microcontroller providing a transmitter disable signal to said transmitter to disable said transmitter when said crypto key and said associated checkword are loaded into said encryption device preventing said crypto key and said associated checkword from being transmitted by said transmitter;

a first light emitting diode connected to said microcontroller, said first light emitting diode displaying a status for a load of said crypto key and said associated checkword into said encryption

device;

said microcontroller being connected to a missile interface within said missile to receive a launch signal from said missile interface upon a launch of said missile, said microcontroller, responsive to said launch signal, erasing said crypto key and said associated checkword from said microcontroller;

a second light emitting diode connected to said microcontroller, said second light emitting diode displaying a status for an erase of said crypto key and said associated checkword from said microcontroller.

14. The apparatus of claim 13 wherein said microcontroller comprises an 8-bit Microcontroller.

15. The apparatus of claim 13 wherein said microcontroller includes an EEPROM for storing said crypto key and said associated checkword and a copy of said crypto key and said associated checkword.

16. The apparatus of claim 13 wherein said microcontroller is connected to a loader interface within

said missile to receive an erase signal from said loader interface, said microcontroller, responsive to said erase signal, erasing said crypto key and said associated checkword from said microcontroller.

ABSTRACT

A Non-Volatile Memory circuit which functions as an interface between a key loader and an encryption device. Included in the Non-Volatile Memory circuit is a microcontroller which has an EEPROM adapted for storage of a crypto key and its corresponding checkword and also a backup crypto key and checkword. Connected to the microcontroller is a 4 MHz clock signal generator which supplies the master clock signal to the microcontroller. A pair of light emitting diodes are also connected to the micrcontroller to indicate the status of a load of the crypto key within the microcontroller as well as the status of an erase of the crypto key from the microcontroller. The microcontroller is also connected to the telemeter transmitter for the missile. This allows the micrcontroller to turn off the transmitter during a key load which prevents transmission of the crypto key and its corresponding checkword. When the microcontroller completes a load of the crypto key from its internal EEPROM to the encryption device and upon a launch of the missile, the software within the microcontroller erases the crypto key and its corresponding checkword from its EEPROM. This prevents an enemy force from retrieving the crypto key and its corresponding checkword from the missile after launch.

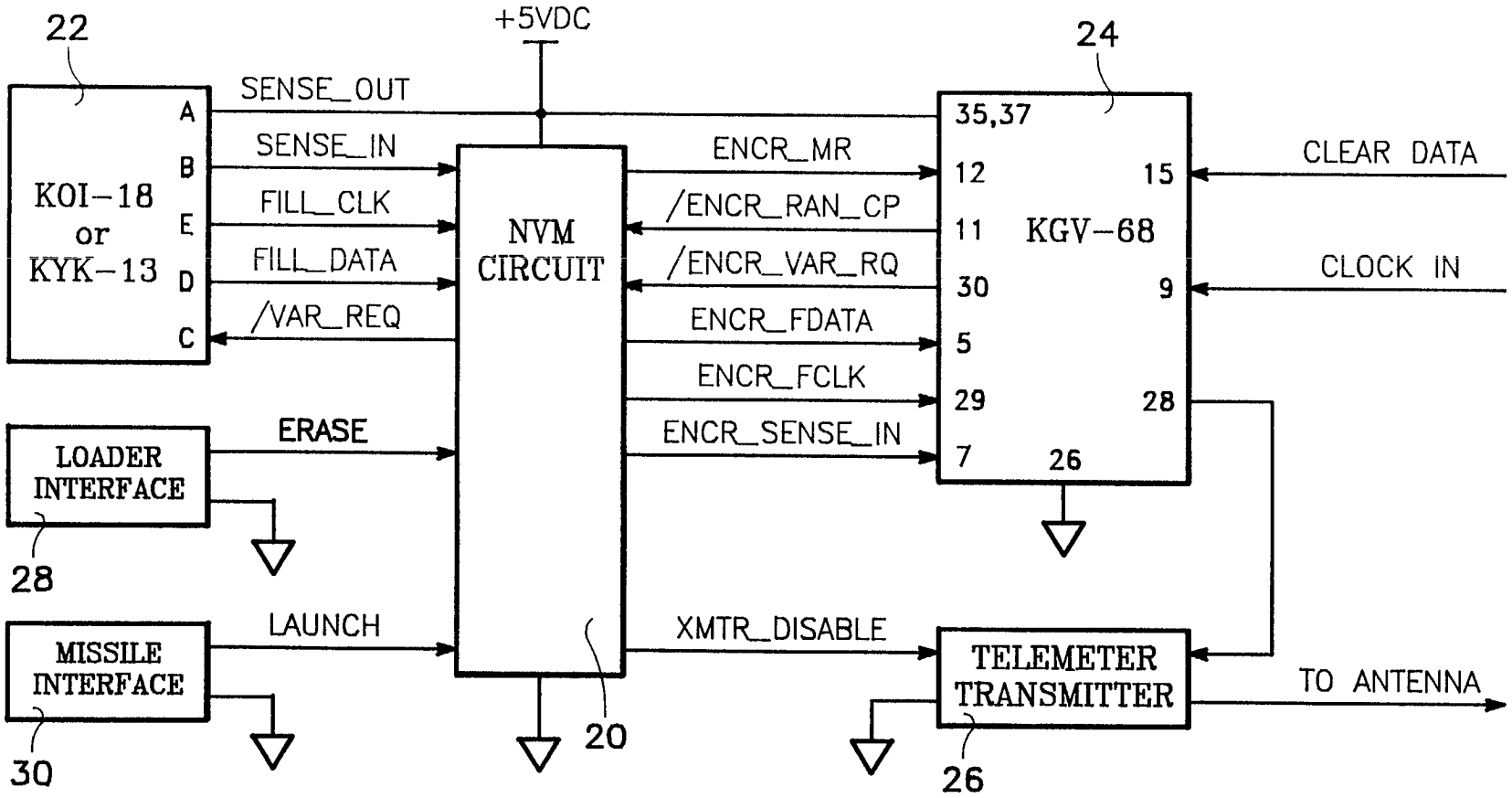

22
+5VDC
24
SENSE_OUT
A
SENSE_IN
B
FILL_CLK
E
FILL_DATA
D
/VAR_REQ
C
KOI-18
or
KYK-13
NVM
CIRCUIT
ENCR_MR
/ENCR_RAN_CP
/ENCR_VAR_RQ
ENCR_FDATA
ENCR_FCLK
ENCR_SENSE_IN
35,37
12
11
30
5
29
7
KGV-68
15
9
28
26
CLEAR DATA
CLOCK IN
ERASE
LOADER
INTERFACE
28
LAUNCH
MISSILE
INTERFACE
30
20
XMTR_DISABLE
TELEMETER
TRANSMITTER
26
TO ANTENNA

FIG. 1

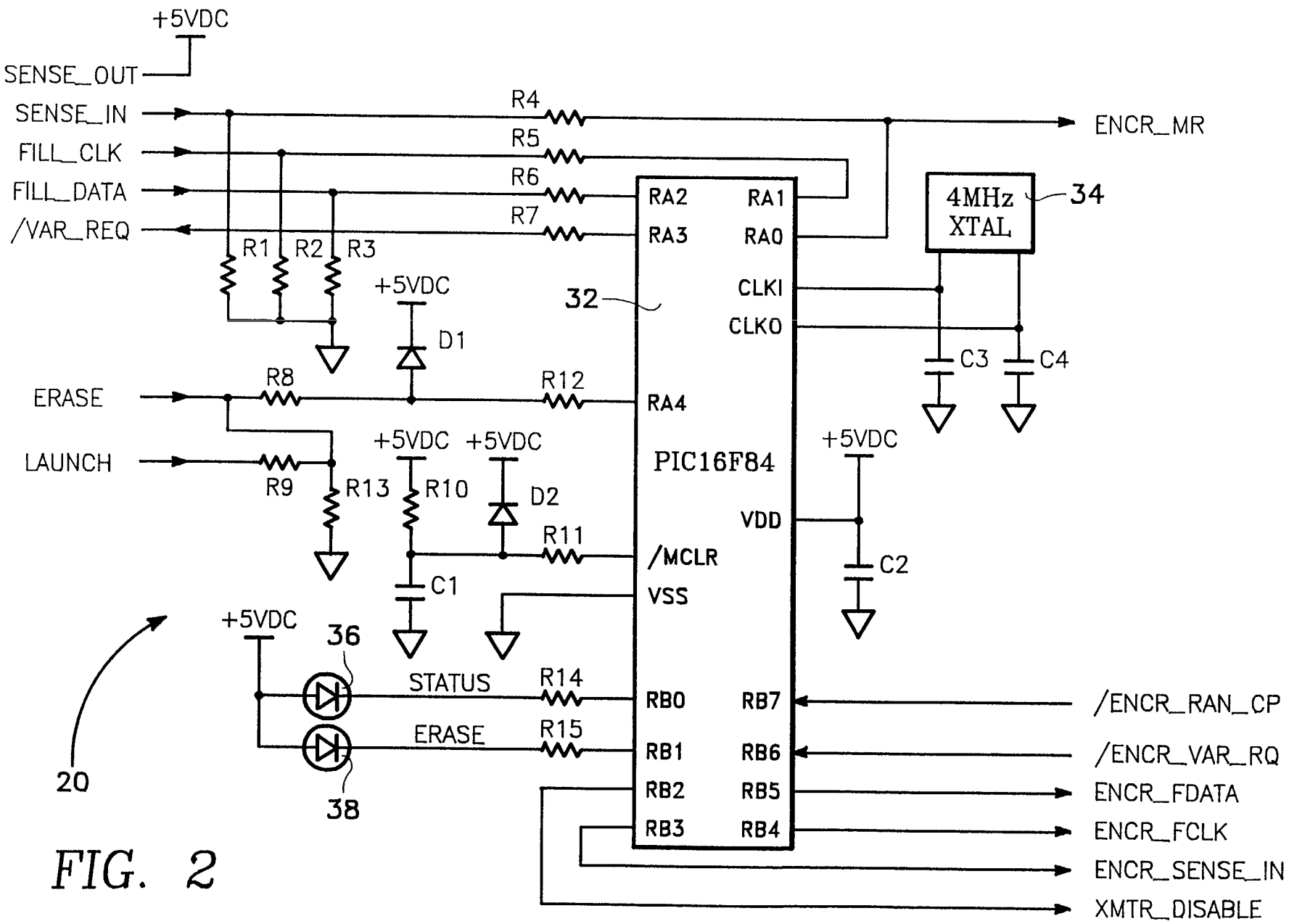

+5VDC
SENSE_OUT
SENSE_IN
FILL_CLK
FILL_DATA
/VAR_REQ
R1
R2
R3
R4
R5
R6
R7
ENCR_MR
4MHz
XTAL
34
32
RA2
RA1
RA3
RA0
CLKI
CLKO
C3
C4
D1
ERASE
R8
R12
RA4
LAUNCH
R9
R13
R10
D2
R11
PIC16F84
VDD
/MCLR
VSS
C1
C2
36
STATUS
ERASE
R14
R15
RB0
RB7
RB1
RB6
RB2
RB5
RB3
RB4
/ENCR_RAN_CP
/ENCR_VAR_RQ
ENCR_FDATA
ENCR_FCLK
ENCR_SENSE_IN
XMTR_DISABLE
20
38

FIG. 2

FIG. 3A

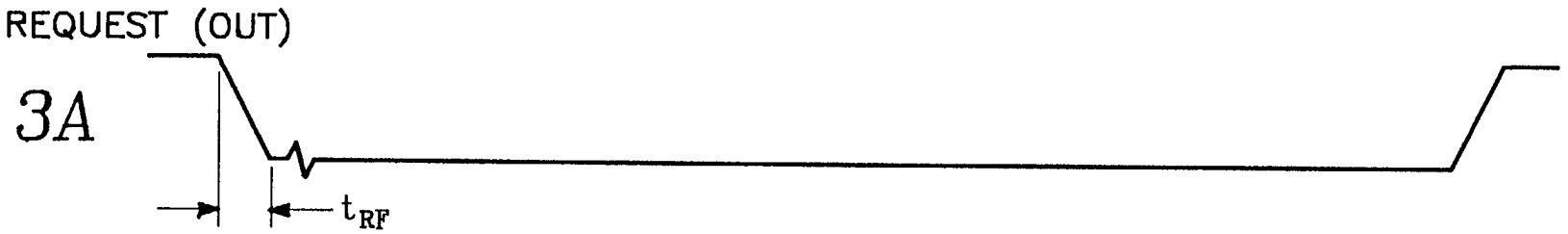


FIG. 3B

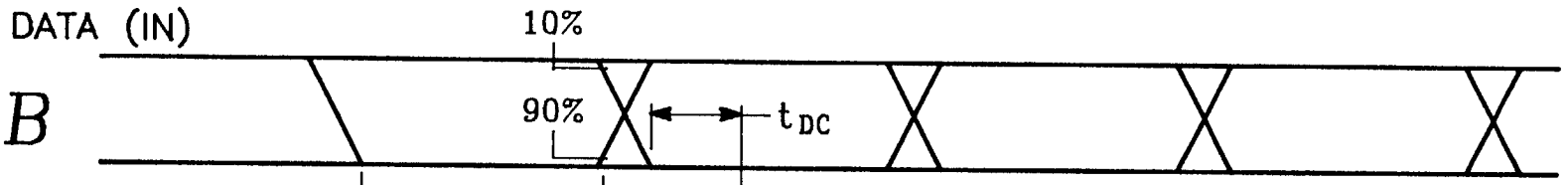
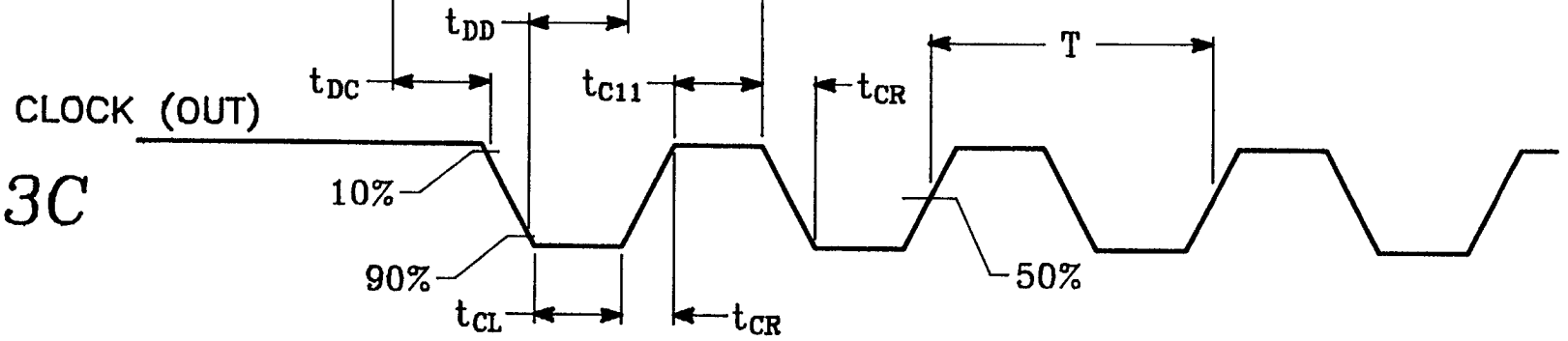


FIG. 3C



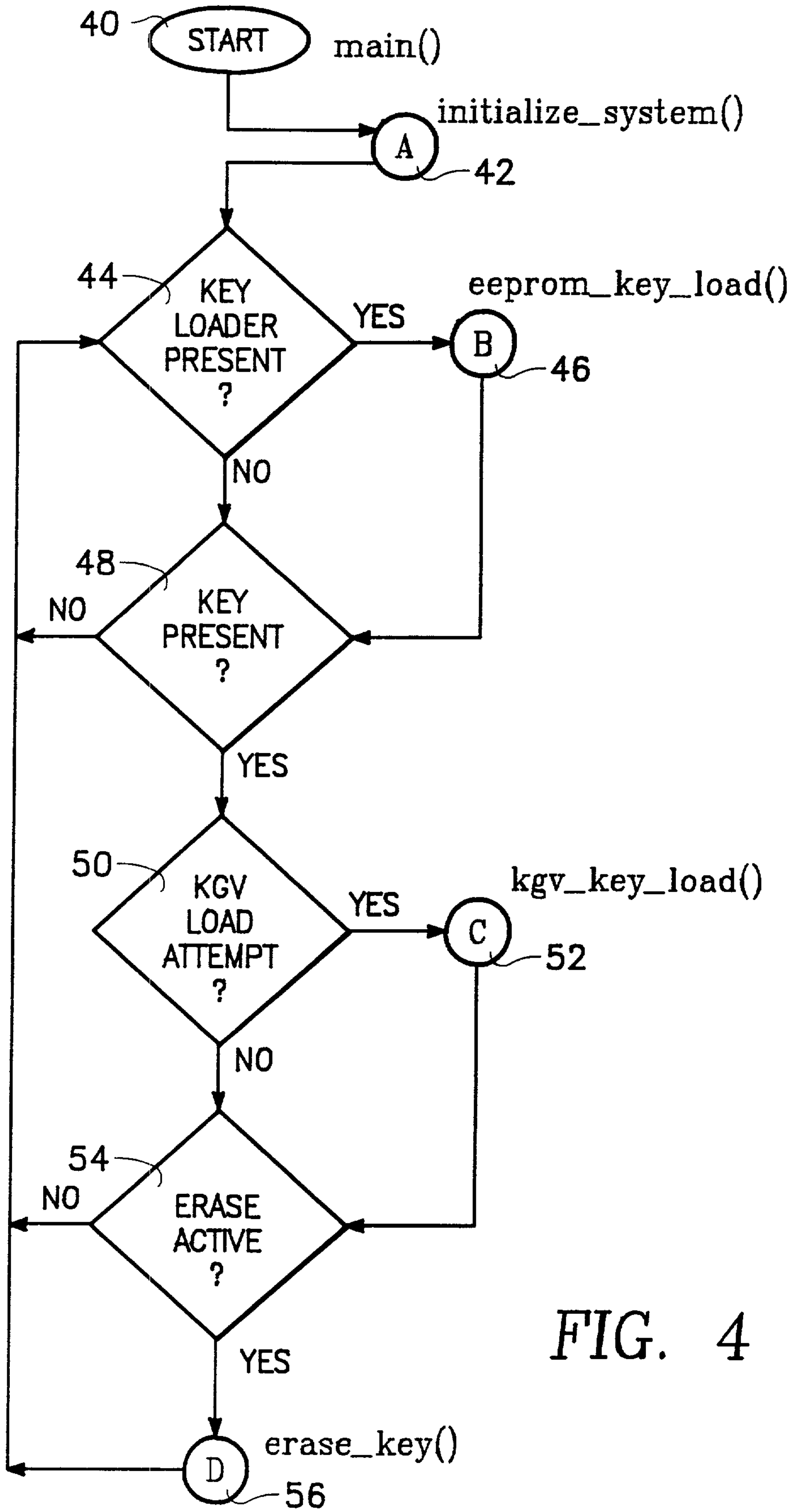


FIG. 4

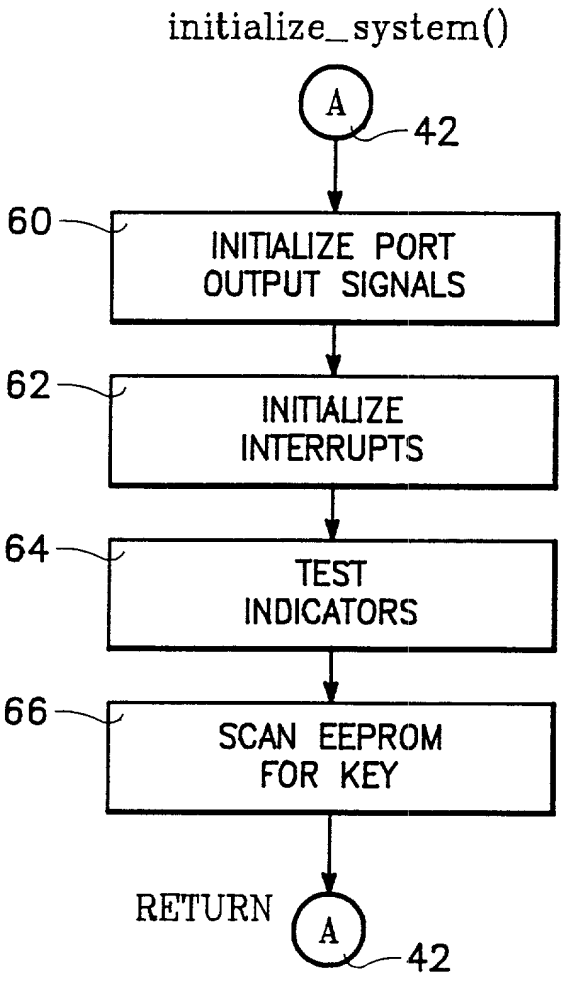


FIG. 5

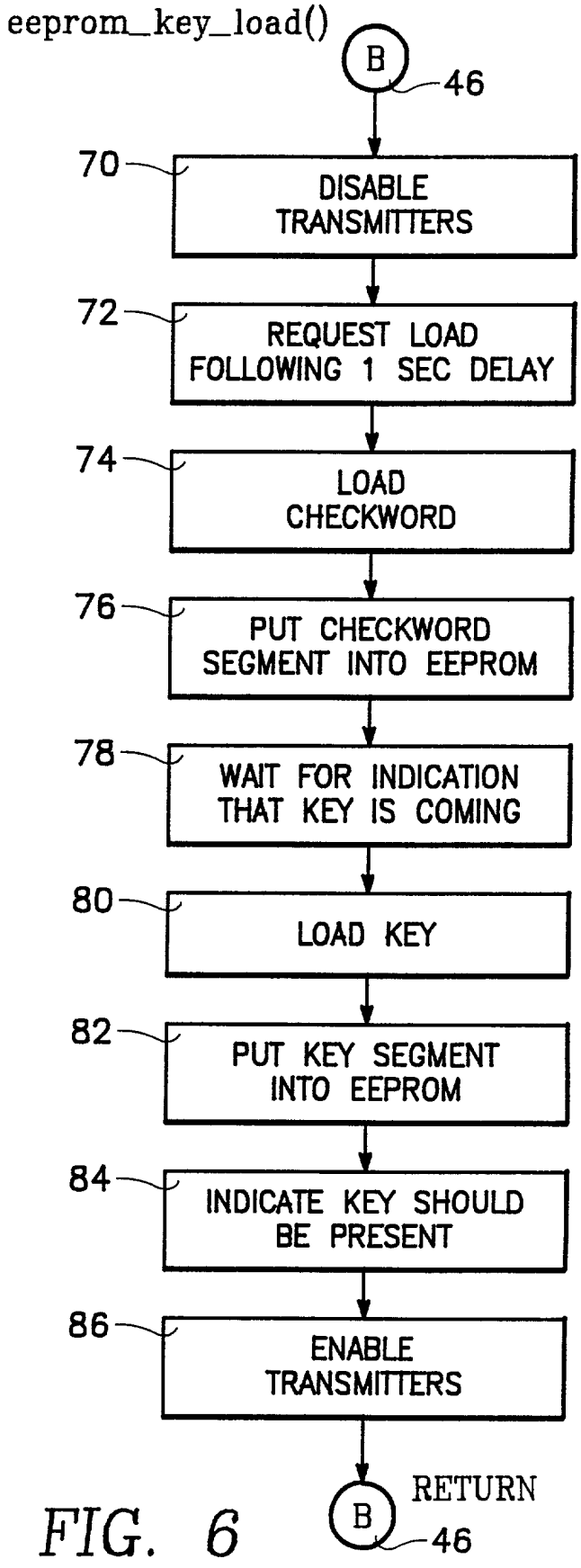


FIG. 6

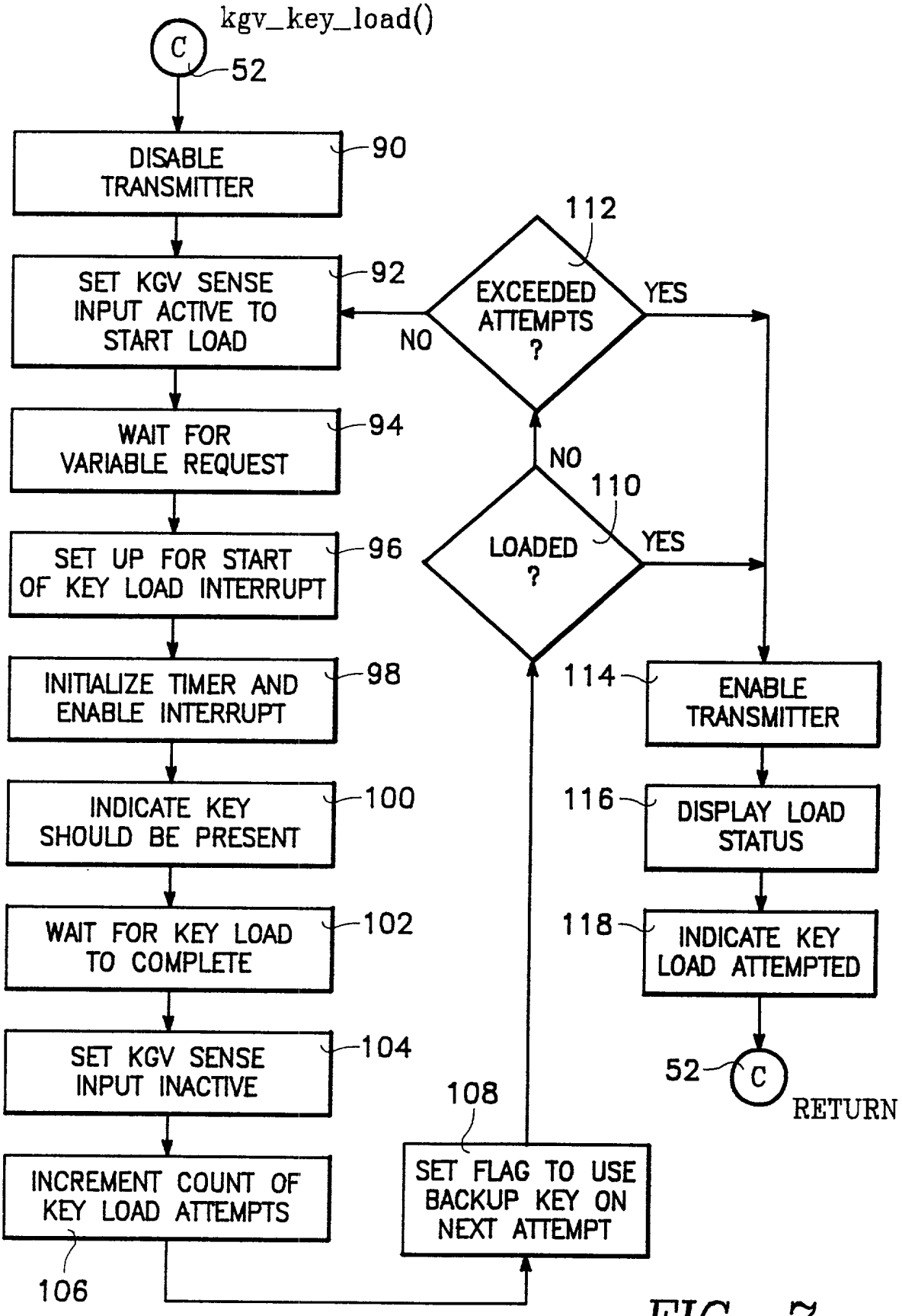

kgv_key_load()
C
52
DISABLE TRANSMITTER
90
SET KGV SENSE INPUT ACTIVE TO START LOAD
92
WAIT FOR VARIABLE REQUEST
94
SET UP FOR START OF KEY LOAD INTERRUPT
96
INITIALIZE TIMER AND ENABLE INTERRUPT
98
INDICATE KEY SHOULD BE PRESENT
100
WAIT FOR KEY LOAD TO COMPLETE
102
SET KGV SENSE INPUT INACTIVE
104
INCREMENT COUNT OF KEY LOAD ATTEMPTS
106
EXCEEDED ATTEMPTS ?
112
NO
YES
LOADED ?
110
NO
YES
SET FLAG TO USE BACKUP KEY ON NEXT ATTEMPT
108
ENABLE TRANSMITTER
114
DISPLAY LOAD STATUS
116
INDICATE KEY LOAD ATTEMPTED
118
52
C
RETURN

FIG. 7

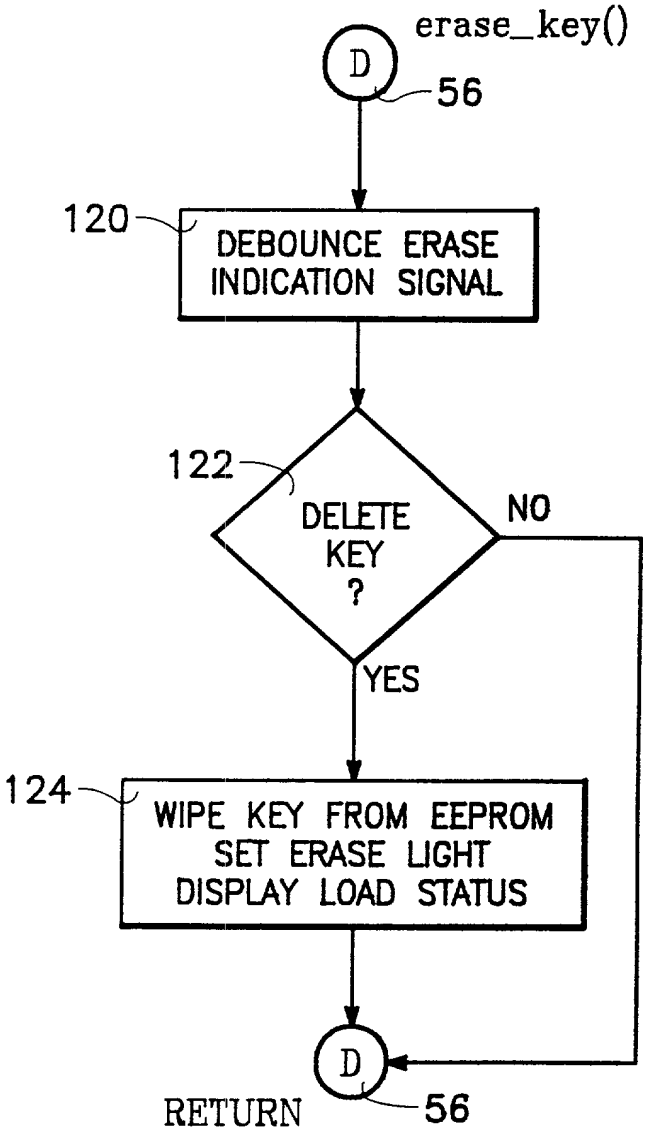


FIG. 8

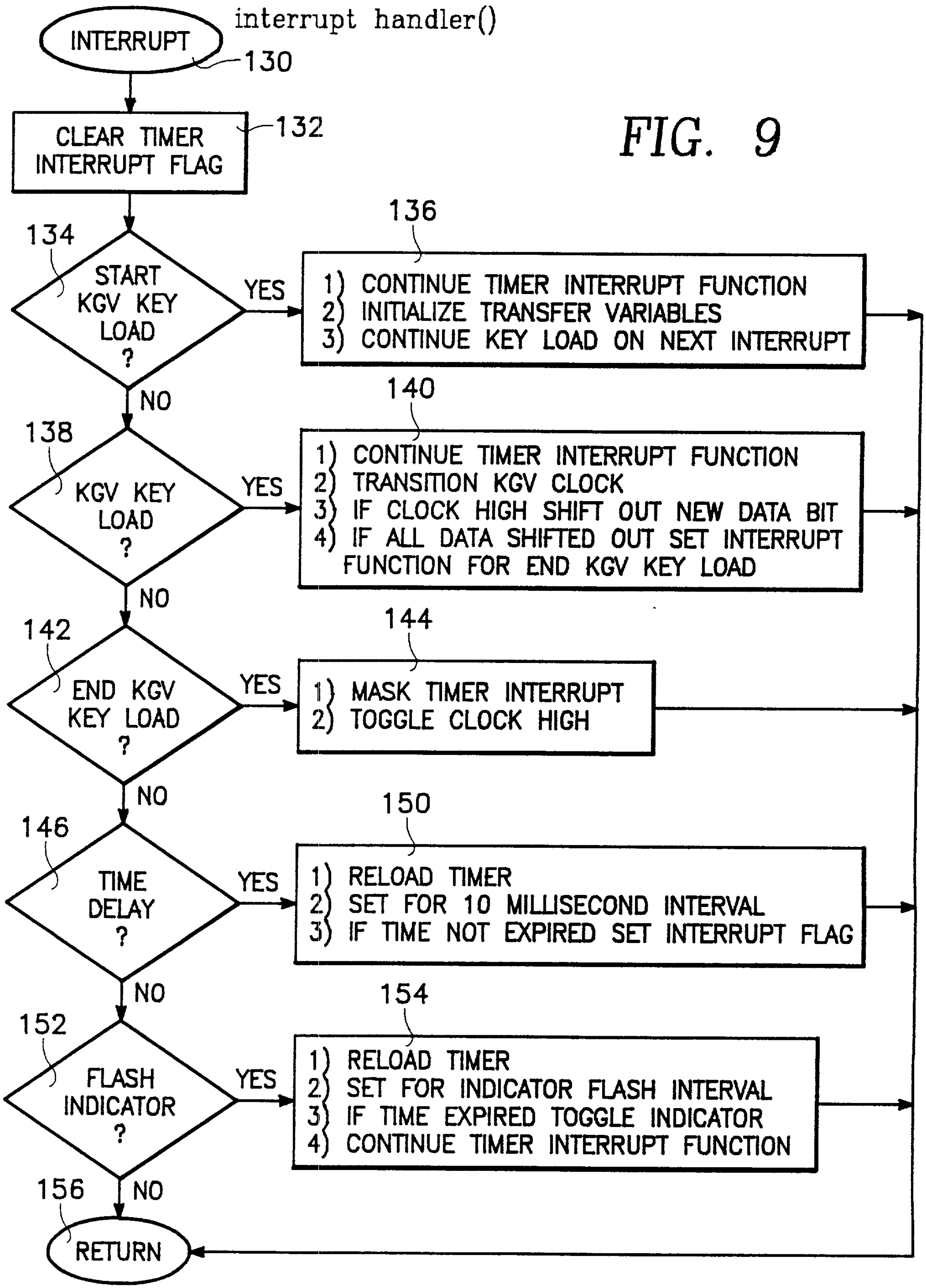


FIG. 9

Navy Case No. **82100**

Declaration and Power of Attorney for Patent Application

As below named inventors, We hereby declare that:

Our residences, post office addresses, and citizenship are as stated below next to our names.
We believe we are the original, first, and joint inventors of the subject matter which is claimed and for which a patent sought on the invention entitled:

NON-VOLATILE MEMORY FOR USE WITH AN ENCRYPTION DEVICE

the specification of which is being filed in this Application.

We hereby state that we have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

We acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, § 1.56(a).

POWER OF ATTORNEY: As the named inventors, We hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith and hereby certify that the Government of the United States has the irrevocable right to prosecute this application:

DAVID S. KALMBAUGH
Registration No. 29234

SEND CORRESPONDENCE TO:
COMMANDER
OFFICE OF COUNSEL, 772000E
NAVAIRWARCENWPNDIV
521 9TH STREET
POINT MUGU, CA 93042-5001

DIRECT TELEPHONE CALLS TO:
DAVID S. KALMBAUGH
Associate Counsel (Intellectual Property)
(805) 989-8266

We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of sole or first inventor CHRISTIAN L. HOULBERG
Residence 9524 ONEIDA STREET, VENTURA, CA 93004 Citizenship: USA
Post Office Address 9524 ONEIDA STREET, VENTURA, CA 93004

Inventor's signature Christian L Houlberg Date 2-8-00

Full name of second inventor GARY S. BORGEN
Residence 5977 SADDLEBACK WAY, CAMARILLO, CA 93012 Citizenship USA
Post Office Address 5977 SADDLEBACK WAY, CAMARILLO, CA 93012

Inventor's signature Gary S. Borgen Date 2-8-00

U.S.P.T.O. Modified Form Apr 85

Appendix A

```
/*************************************************************************

      Configuration Bits

*************************************************************************/

      #define _CP_ON           0x000F
      #define _CP_OFF          0x3FFF
      #define _PWRTE_ON        0x3FF7
      #define _PWRTE_OFF       0x3FFF
      #define _WDT_ON          0x3FFF
      #define _WDT_OFF         0x3FFB
      #define _LP_OSC          0x3FFC
      #define _XT_OSC          0x3FFD
      #define _HS_OSC          0x3FFE
      #define _RC_OSC          0x3FFF

      #define       ___mkstr(x) #x
#ifdef PIC_PROGRAMMER
      #define       __CONFIG(x) asm("\tpsect config,class=CODE,delta=2"); \
                                asm("\tglobal\tconfig_word"); \
                                asm("config_word"); \
                                asm("\tdw "___mkstr(x))
#endif      /* End of PIC_PROGRAMMER */
#ifdef DATAIO_PROGRAMMER      /* Locate configuration at 0x0404 */
      #define       __CONFIG(x) asm("\tpsect dataio,class=CODE,delta=2"); \
                                asm("\tglobal\tconfig_word"); \
                                asm("config_word"); \
                                asm("\tdw "___mkstr(x))
#endif      /* End of DATAIO_PROGRAMMER */


/* end */

//    psect eedata,delta=2,abs,ovrld
//    org         2100h
//    db          1,2,3,4,5
```

```
/*****************************************************************************
Module Name:      nvmem.c
Number/Version:   1.00
History:
      Date              Rev         Author                  Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Functions:
      initialize_system()           Initializes processor.
      eeprom_key_load()             Loads key from key loader and stores it in
                                    EEPROM.
      kgv_key_load()                Loads the key stored in the EEPROM into the
                                    KGV-68.
      erase_key()                   Erases key following an erase indication.
      wipe_key()                    Wipe the key from EEPROM memory.
      time_delay()                  Time delay (sets up interrupt routine).

Abstract:
    This program performs all Non-Volital Memory control functions.
     The PIC16F873 or PIC16F876 is used as the Non-Volital Memory device.
     The device signal definitions follow:
     Key Loader Data Interface Signals
1)         sense_in          Digital input          Signal activating KGV-68 for
                                                    keying
1)         fill_clk          Digital input          Non-volatile memory key load clock
1)         fill_data         Digital input          Non-volatile memory key load data
1)         var_req           Digital output         Strobe requesting key load
1)         erase             Discrete input         Analog input 2.5 Volt threshold
     Key Loader Indicator Signals
1)         kgv1_ok           Digital output         KGV1 key load accepted and OK
1)         erase_ind         Digital output         Erased key indicator
           kgv2_ok           Digital output         KGV2 key load accepted and OK
     System Interface Signals
           flight_erase      Discrete input         Analog input 22.5 Volt threshold
1)         xmtr_disable      Digital output         Transmitter disable signal
     KGV Interface Signals
1)         encr_sen_in1      Digital output         Sense signal for KGV1
1)         encr_fclk         Digital output         KGV key loading clock (1.6KHz)
1)         encr_fdata        Digital output         KGV key loading data
1)         encr_var_req      Digital input          KGV key variable request
1)         encr_ran_cp1      Digital input          KGV1 random compare OK (active
                                                    low)
           encr_sen_in2      Digital output         Sense signal for KGV2
           encr_mr           Digital output         KGV master reset
           encr_ck_ok1       Digital input          KGV1 key check OK (active low)
           encr_ck_ok2       Digital input          KGV2 key check OK (active low)
           encr_ran_cp2      Digital input          KGV2 random compare OK (active
                                                    low)

Notes:
      1)    Minimal set up I/O signals required to perform the non-volital
            memory function (single KGV-68).  The non-volatile memory function
            can therefore be implemented with a PIC16F83 microcontroller.
      2)    A PIC16F876 is used to perform the non-volital memory function for
            applications requiring two KGV-68s.  All the above signals are
            used in this implementation.
      3)    The processor is operated with a clock rate of 4MHz.  All timer
```

```
            operations must adjust prescaler and counter registers accordingly.
      4)    All timer operations are implemented with an interrupt.  Global
            variables are used to identify the timer function.
*****************************************************************************/
/*    Conditional compilation.
*/
//    #define DUAL_KGV_SYSTEM                    /* Two KGV-68s to load */
      #define MAX_KEYLOAD_ATTEMPTS  3            /* Attempts for each key copy */

/*    Programmer being used.
*/
      /* For the PIC programmer, no user defined memory section needed */
//    #define PIC_PROGRAMMER
      /* For the DataIO, the PICC command line must include -l-pdataio=0404h */
      #define DATAIO_PROGRAMMER

/*    All parameters and functions used by main() are defined in
      the following header files.
*/
#ifdef DUAL_KGV_SYSTEM
   #include <pic16876.h>
#else                                 /* Single KGV system */
   #include <pic1684.h>               /* Processor definitions */
#endif                                /* End of KGV system declaration */
   #include "config.h"                /* PIC configuration definitions */
   #include "nvmem.h"                 /* Non-Volital Memory control definitions */
   #include "size.h"                  /* Sensitive data size information */

/*    Configuration.
*/
   __CONFIG(_CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC);

/*    Constant definitions.
*/
   #define DEGLICH_COUNT         3          /* Consecutive active signal
                                               samples */

/*    Data storage locations.
*/
      #define PRIMARY_CW_STORAGE    0x00
      #define PRIMARY_KEY_STORAGE   (PRIMARY_CW_STORAGE + CHECK_WORD_SIZE)
      #define BACKUP_CW_STORAGE     (PRIMARY_KEY_STORAGE + KEY_SIZE)
      #define BACKUP_KEY_STORAGE    (BACKUP_CW_STORAGE + CHECK_WORD_SIZE)
      #define TOTAL_KEY_STORAGE     ((CHECK_WORD_SIZE + KEY_SIZE) << 1)

/*    Enumerations.
*/
      enum Activation
      {
            OFF,
            ON
      };

      enum Encrypter
      {
            KGV1,
            KGV2
```

```
};

enum KeySource
{
      BACKUP,
      PRIMARY
};

enum InterruptFunction
{
      START_KGV_KEY_LOAD,
      KGV_KEY_LOAD,
      END_KGV_KEY_LOAD,
      TIME_DELAY,
      FAST_FLASH,
      SLOW_FLASH
};
/* Global interrupt enable */
#define GLOBAL_ENABLE               0x80

/* (304 * 4 = 1,216 clock cycle half period => 1,645 Hz) */
/* Internal clock, low to high, prescale 1/16 */
#define KGV_KEY_LOAD_OPTION         0x03
/* 1/19 (1/16 * 1/19 = 1/304) */
#define KGV_KEY_LOAD_TMR0           (256 - 19)
/* Internal clock, low to high, prescale 1/256 */
#define TEN_MSEC_TIMER_OPTION 0x07
/* 1/39 (1/256 * 1/39 = 1/9,984) */
#define TEN_MSEC_TIMER_TMR0         (256 - 39)
/* Fast flash is 10 flashes/second slow flash is 2 flashes/second */
#define INDICATOR_FLASH_OPTION      0x07
#define INDICATOR_FLASH_TMR0  (256 - 39)
#define FAST_FLASH_COUNT            5
#define SLOW_FLASH_COUNT            25

/*  Signal declarations.
*/
#define key_loader_present    sense_in
#define kgv1_not_loaded       encr_ran_cp1
#define kgv2_not_loaded       encr_ran_cp2
#define launch_active         flight_erase
#define erase_active          erase

/*    Variable declarations.
*/
      /* Source of key for key load and KGV being loaded */
      unsigned char key_source = PRIMARY;
      unsigned char key_destination = KGV1;
      /* Key load attempts */
      unsigned char kgv1_load_attempt = 0;
      unsigned char kgv2_load_attempt = 0;
      unsigned char kgv_load_attempted = 0;
      /* Storage for timer function interrupt */
      unsigned char key_addr;
      unsigned char key_byte;
      unsigned char shift_counter;
      /* Interrupt function */
```

```
      unsigned char interrupt_function;
      /* Global timer count down */
      unsigned char timer_count;
      unsigned char fudge_count;

/*    Function definitions.
*/
      void initialize_system(unsigned char *key_present_ptr);
      void eeprom_key_load(unsigned char *key_present_ptr);
      unsigned char get_byte(void);
      void kgv_key_load(void);
      void interrupt handler(void);
      unsigned char read_eeprom(unsigned char address);
      void erase_key(void);
      void wipe_key(void);
      void time_delay(void);
      void check_eeprom(unsigned char *key_present_ptr);
      void display_load_status(void);

/*****************************************************************************
Function Name:    main()
Number/Version:
History:
   Date              Rev         Author                    Description
   17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
   None.

Output Variables:
   None.

Global Variables:
   None.

Functions Referenced:
      initialize_system()          Initializes processor.
      eeprom_key_load()       Loads key from key loader and stores it in EEPROM.
      kgv_key_load()               Loads key into KGVs.  Transmitters off during load.
      erase_key()             Erases key following an erase indication.

Abstract:   Main program module to perform all Non-Volital Memory Control
      functions.
*****************************************************************************/
void main(void)
{
      /* Variable declarations */
      unsigned char key_present;

      /* Initialize the system */
      initialize_system(&key_present);

      /* Key loading */
      for(;;)
      {
            /* Check if loader is present (returns when not present) */
            if(key_loader_present)
```

```
                eeprom_key_load(&key_present);

            /* Check if key is present */
            if(key_present)
            {
                /* Load the key into the KGVs */
                if(!kgv_load_attempted)       /* Only if not previously
                                                 attempted */
                    kgv_key_load();

                /* Check for erase indication to erase key */
                if(erase_active)
                    erase_key();
            }
        }
}

/*****************************************************************************
Function Name:    initialize_system()
Number/Version:
History:
    Date              Rev         Author                    Description
    17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
    key_present_ptr             Pointer to key_present flag.

Output Variables:
    None.

Global Variables:
    None.

Functions Referenced:
    time_delay()                Wait for timer count to expire.
    eeprom_read()               Get byte from EEPROM - PIC library function.

Abstract:   Initializes system for all Non-Volital Memory Control functions.
*****************************************************************************/
void initialize_system(unsigned char *key_present_ptr)
{
    /* Initialize port data direction */
    TRISA = PORT_A_DIRECTION;
    TRISB = PORT_B_DIRECTION;
#ifdef DUAL_KGV_SYSTEM
    TRISC = PORT_C_DIRECTION;
#endif

    /* Initialize port output signal levels */
    var_req = !0;                       /* Active low (not requesting load) */
    kgv1_ok = !0;                       /* Active low (KGV1 not loaded) */
    erase_ind = !0;             /* Active low (key not erased) */
    xmtr_disable = !0;                  /* Active high (transmitter disabled) */
    encr_sen_in1 = 0;           /* Active high (not loading KGV1) */
    encr_fclk = !0;             /* Active falling edge (initially high) */
    encr_fdata = !0;            /* Zero data bit */
#ifdef DUAL_KGV_SYSTEM
```

```
      kgv2_ok = !0;                        /* Active low (KGV2 not loaded) */
      encr_sen_in2 = 0;                 /* Active high (not loading KGV2) */
      encr_mr = 0;                         /* Active high (not performing reset) */
#endif

      /* Initialize interrupts */
      INTCON = GLOBAL_ENABLE;          /* Global enable, mask all interrupts */

      /* Test indicators */
      kgv1_ok = 0;                         /* Indicator on */
      timer_count = 100;                   /* 1 second interval */
      time_delay();                        /* Delay for indicated count */
      kgv1_ok = !0;                        /* Indicator off */
#ifdef DUAL_KGV_SYSTEM
      kgv2_ok = 0;                         /* Indicator on */
      timer_count = 100;                   /* 1 second interval */
      time_delay();                        /* Delay for indicated count */
      kgv2_ok = !0;                        /* Indicator off */
#endif
      erase_ind = 0;                       /* Indicator on */
      timer_count = 100;                   /* 1 second interval */
      time_delay();                        /* Delay for indicated count */
      erase_ind = !0;                   /* Indicator off */

      /* Scan EEPROM for presence of key */
      check_eeprom(key_present_ptr);
}

/*****************************************************************************
Function Name:    eeprom_key_load()
Number/Version:
History:
      Date              Rev         Author                    Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      key_present_ptr               Pointer to key_present flag.

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      time_delay()                  Wait for timer count to expire.
      get_byte()                 Get byte from KYK-13 or KOI-18.
      eeprom_write()                Put byte in EEPROM - PIC library function.

Abstract:   Loads the check word and key from key loader (KYK-13 or KOI-18)
      and stores it in EEPROM.
*****************************************************************************/
void eeprom_key_load(unsigned char *key_present_ptr)
{
      /* Variable declarations */
      unsigned char byte_count, stable_count;
      unsigned char key_segment[16];       /* Temporary storage for key segment */
```

```
      /* Disable the transmitters */
      xmtr_disable = !0;

      /* Request load after an approximate 1 second delay */
      kgv1_ok = !0;                              /* Ensure indicator flash off */
#ifdef DUAL_KGV_SYSTEM
      kgv2_ok = !0;                              /* Indicator off */
#endif
      timer_count = 10;                    /* 0.1 second interval */
      time_delay();                              /* Delay for indicated count */
      var_req = 0;                               /* Active low request */

      /* Load Check Word */
      for(byte_count = 0;
            (byte_count < CHECK_WORD_SIZE) && key_loader_present; byte_count++)
      {
            /* Get one byte of fill data */
            key_segment[byte_count] = get_byte();

            /* Set request inactive (arbitrarily set done after 1st byte) */
            var_req = !0;           /* Active low request now not active */
      }

      /* Put Check Word segment into EEPROM */
      for(byte_count = 0;
            (byte_count < CHECK_WORD_SIZE) && key_loader_present; byte_count++)
      {
            eeprom_write(PRIMARY_CW_STORAGE + byte_count, key_segment[byte_count]);
            eeprom_write(BACKUP_CW_STORAGE + byte_count, key_segment[byte_count]);
      }

      /* Wait for indication that key is coming */
      if(key_loader_present)
      {
            /* Wait for disconnect (KYK-13) or fill clock (KOI-18) */
            while(key_loader_present && fill_clk);

            /* Check for disconnect */
            if(!key_loader_present) /* Loading with KYK-13 */
            {
                  /* Wait for key loader present */
                  for(stable_count = 0; (stable_count < DEGLICH_COUNT);
                        stable_count++)
                        if(!key_loader_present) stable_count = 0;

                  /* Set request inactive (arbitrarily set done after 1st byte) */
                  var_req = 0;            /* Active low request now active */
            }
      }

      /* Load Key */
      for(byte_count = 0;
            (byte_count < KEY_SIZE) && key_loader_present; byte_count++)
      {
            /* Get one byte of fill data */
            key_segment[byte_count] = get_byte();
```

```
            /* Set request inactive (arbitrarily set done after 1st byte) */
            var_req = !0;           /* Active low request now not active */
      }

      /* Put Key segment into EEPROM */
      for(byte_count = 0;
            (byte_count < KEY_SIZE) && key_loader_present; byte_count++)
      {
            eeprom_write(PRIMARY_KEY_STORAGE + byte_count, key_segment[byte_count]);
            eeprom_write(BACKUP_KEY_STORAGE + byte_count, key_segment[byte_count]);
      }

      /* Indicate key should be present (procedure completed) */
      if(key_loader_present)
      {
            *key_present_ptr = !0;

            /* Clear erase light */
            erase_ind = !0;

            /* Wait for loader to be disconnected or turned off */
            while(key_loader_present);

            /* Indicate key load should be attempted */
            kgv_load_attempted = 0;
            kgv1_load_attempt = 0;
#ifdef DUAL_KGV_SYSTEM
            kgv2_load_attempt = 0;
#endif
      }
      else
      {
            /* Check EEPROM for old key */
            check_eeprom(key_present_ptr);

            /* Display load status */
            display_load_status();
      }

      /* Enable the transmitters */
      xmtr_disable = 0;
}

/*****************************************************************************
Function Name:    kgv_key_load()
Number/Version:
History:
      Date              Rev         Author                    Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      None.

Global Variables:
```

```
      None.

Functions Referenced:
      None.

Abstract:   Loads the key in EEPROM into the KGV-68s.  The transmitters are
      disabled during the key load process.  This function can be compiled
      for optimal operation with one or two KGV-68s.
*****************************************************************************/
void kgv_key_load(void)
{
      /* Disable the transmitters */
      xmtr_disable = !0;

      /* Attempt key load until maximum attempts are exceeded */
      do
      {     /* Check if KGV1 is not loaded */
            if(kgv1_not_loaded)
            {
                  /* Set KGV1 sense input active to start load */
                  encr_sen_in1 = !0;

                  /* Wait for variable request from KGV1 */
                  while(encr_var_req);          /* Active low (wait for low) */

                  /* Attempt a key load */
                  /* Set up for start of key load interrupt */
                  interrupt_function = START_KGV_KEY_LOAD;
                  key_destination = KGV1;

                  /* Initialize timer and enable interrupt */
                  TMR0 = KGV_KEY_LOAD_TMR0;
                  OPTION = KGV_KEY_LOAD_OPTION;
                  INTCON = GLOBAL_ENABLE;
                  T0IE = 1;

                  /* Wait for key load to complete */
                  while(T0IE == 1);

                  /* Set KGV1 sense input inactive */
                  encr_sen_in1 = 0;

                  /* Count key load attempts */
                  ++kgv1_load_attempt;
            }

#ifdef DUAL_KGV_SYSTEM
            /* Check if KGV2 is not loaded */
            if(kgv2_not_loaded)
            {
                  /* Set KGV2 sense input active to start load */
                  encr_sen_in2 = !0;

                  /* Wait for variable request from KGV2 */
                  while(encr_var_req);          /* Active low (wait for high) */

                  /* Attempt a key load */
```

```
                /* Set up for start of key load interrupt */
                interrupt_function = START_KGV_KEY_LOAD;
                key_destination = KGV2;

                /* Initialize timer and enable interrupt */
                TMR0 = KGV_KEY_LOAD_TMR0;
                OPTION = KGV_KEY_LOAD_OPTION;
                INTCON = GLOBAL_ENABLE;
                T0IE = 1;

                /* Wait for key load to complete */
                while(T0IE == 1);

                /* Set KGV2 sense input inactive */
                encr_sen_in2 = 0;

                /* Count key load attempts */
                ++kgv2_load_attempt;
            }
#endif
            /* Next try other key source */
            key_source = !key_source;

            /* Delay to allow the KGV-68 time to process segment */
            timer_count = 100;
            time_delay();
        }
        /* Alternate between the primary key and the backup key */
        while((kgv1_not_loaded
            && (kgv1_load_attempt < (MAX_KEYLOAD_ATTEMPTS << 1)))
#ifdef DUAL_KGV_SYSTEM
            || (kgv2_not_loaded
            && (kgv2_load_attempt < (MAX_KEYLOAD_ATTEMPTS << 1)))
#endif
        );

        /* Enable the transmitters */
        xmtr_disable = 0;

        /* Display indication if properly loaded */
        display_load_status();

        /* Indicate KGV load attempted */
        kgv_load_attempted = !0;
}

/*****************************************************************************
Function Name:    handler()
Number/Version:
History:
        Date              Rev         Author                    Description
        17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
        None.

Output Variables:
```

```
      None.

Global Variables:
      None.

Functions Referenced:
      eeprom_read()                 Get byte from EEPROM - PIC library function.

Abstract:   Loads the key in EEPROM into the KGV-68.
*****************************************************************************/
void interrupt handler(void)
{
      /* Variable declarations */
      unsigned char temp;

      /* Clear TMR0 flag */
      T0IF = 0;

      /* Determine function of interrupt */
      switch(interrupt_function)
      {
            case(START_KGV_KEY_LOAD):
                  /* Continue timer interrupt key load function */
                  TMR0 = KGV_KEY_LOAD_TMR0;
                  OPTION = KGV_KEY_LOAD_OPTION;
                  T0IE = 1;

                  /* Initialize transfer variables */
                  encr_fclk = !0;               /* Active falling edge */
                  if(key_source == PRIMARY)    /* Point to start of primary key */
                        key_addr = PRIMARY_CW_STORAGE;
                  else  /* Point to start of backup key */
                        key_addr = BACKUP_CW_STORAGE;

                  key_byte = read_eeprom(key_addr++); /* Get first byte */
                  if(key_byte & 0x80)                       /* Determine state of MSB */
                        encr_fdata = 1;                     /* Output bit */
                  else
                        encr_fdata = 0;          * Output bit */
                  key_byte = key_byte << 1;     /* Shift for next bit transfer */
                  shift_counter = 1;                  /* Indicate first bit output */

               /* New interrupt function (continue key load on next interrupt) */
                  interrupt_function = KGV_KEY_LOAD;
                  break;
            case(KGV_KEY_LOAD):
                  /* Continue timer interrupt key load function */
                  TMR0 = KGV_KEY_LOAD_TMR0;
                  OPTION = KGV_KEY_LOAD_OPTION;
                  T0IE = 1;

                  /* Transition clock */
                  temp = encr_fclk;
                  encr_fclk = !temp;

                  /* Shift out new data bit on falling edge of encr_fclk */
                  if(encr_fclk)
```

```
        {
              /* Check if starting a new byte */
              if(!shift_counter)
                    key_byte = read_eeprom(key_addr++);

              /* Check value of MSB and output to KGV */
              if(key_byte & 0x80)
                    encr_fdata = 1;
              else
                    encr_fdata = 0;

              /* Shift data and updata shift counter modulo 8 */
              key_byte = key_byte << 1;     /* Shift for next bit
                                               transfer */
              shift_counter = ++shift_counter & 0x07;
        }

        /* Check for new timer function */
        if(((key_addr == (PRIMARY_CW_STORAGE + TOTAL_KEY_STORAGE))
              || (key_addr == (BACKUP_CW_STORAGE + TOTAL_KEY_STORAGE)))
              && !shift_counter && !encr_fclk)
              interrupt_function = END_KGV_KEY_LOAD;
        break;
  case(END_KGV_KEY_LOAD):
        /* Mask timer interrupt (key load function completed) */
        T0IE = 0;

        /* Toggle clock high */
        encr_fclk = !0;
        break;
  case(TIME_DELAY):
        /* Reload TMR0 */
        TMR0 = TEN_MSEC_TIMER_TMR0;
        OPTION = TEN_MSEC_TIMER_OPTION;
        if(timer_count)
        {
              --timer_count;
              T0IE = 1;
        }
        else
        {
              T0IE = 0;
        }
        break;
  case(FAST_FLASH):
  case(SLOW_FLASH):
        /* Reload TMR0 */
        TMR0 = INDICATOR_FLASH_TMR0;
        OPTION = INDICATOR_FLASH_OPTION;
        if(timer_count)
        {
              --timer_count;
        }
        else
        {
              /* Re-establish count */
              if(interrupt_function == FAST_FLASH)
```

```
                        timer_count = FAST_FLASH_COUNT;
                    else
                        timer_count = SLOW_FLASH_COUNT;
                    /* Toggle indicator */
                    temp = kgv1_ok;
                    kgv1_ok = !temp;
#ifdef DUAL_KGV_SYSTEM
                    temp = kgv2_ok;
                    kgv2_ok = !temp;
#endif
                }
                T0IE = 1;
                break;
            default:
                break;
        }
}

/*****************************************************************************
Function Name:    read_eeprom()
Number/Version:
History:
    Date              Rev         Author                  Description
    17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
    unsigned char address   EEPROM data address location.

Output Variables:
    unsigned char read_eeprom()         Data from EEPROM.

Global Variables:
    None.

Functions Referenced:
    None.

Abstract:   EEPROM read routine ONLY for interrupt handler routine.
*****************************************************************************/
unsigned char read_eeprom(unsigned char address)
{
    EEADR = address;
    RD = 1;
    return EEDATA;
}

/*****************************************************************************
Function Name:    erase_key()
Number/Version:
History:
    Date              Rev         Author                  Description
    17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
    None.

Output Variables:
```

```
      None.

Global Variables:
      None.

Functions Referenced:
      wipe_key()                    Perform a wipe operation to erase the key.

Abstract:   Erases the key stored in EEPROM following an erase indication
      from the key loader.
*****************************************************************************/
void erase_key(void)
{
      /* Variable declarations */
      unsigned char stable_count;
      unsigned char delete_key = !0;

      /* Debounce the erase indication signal */
      for(stable_count = 0; stable_count < DEGLICH_COUNT; stable_count++)
            if(!erase_active) delete_key = 0;

      /* If indicated, delete the key */
      if(delete_key)
      {
            /* Wipe the key from EEPROM memory */
            wipe_key();

            /* Set erase light when key is erased */
            erase_ind = 0;

            /* Maintain KGV-68 load status */
            display_load_status();
      }
}

/*****************************************************************************
Function Name:    wipe_key()
Number/Version:
History:
      Date              Rev         Author                  Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      eeprom_read()                 Get byte from EEPROM - PIC library function.

Abstract:   Performs a "wipe" operation to erase the key stored in EEPROM.
*****************************************************************************/
void wipe_key(void)
```

```
{
      /* Variable declarations */
      unsigned char key_erased = 0;
      unsigned char erase_pass;
      unsigned char byte_count;
      unsigned char data_byte[] = {0xaa, 0x55, 0x46, 0xff, 0x00};

      while(!key_erased)
      {
            /* Erase EEPROM key storage memory (5 passes) */
            for(erase_pass = 0; erase_pass < 5; erase_pass++)
            {
                  /* Perform one erasure pass */
                  for(byte_count = 0; byte_count < TOTAL_KEY_STORAGE; byte_count++)
                        eeprom_write(PRIMARY_CW_STORAGE + byte_count,
                              data_byte[erase_pass]);
            }

            /* Read EEPROM to verify erasure */
            key_erased = !0;                          /* Assume key was erased */
            for(byte_count = 0; byte_count < TOTAL_KEY_STORAGE; byte_count++)
                  if(eeprom_read(PRIMARY_CW_STORAGE + byte_count))
                        key_erased = 0;
      }
}

/*****************************************************************************
Function Name:    time_delay()
Number/Version:
History:
      Date              Rev         Author                  Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      None.

Abstract:   Set up timer counter and waits until interrupts are completed.
*****************************************************************************/
void time_delay(void)
{
      interrupt_function = TIME_DELAY;/* Set up interrupt */
      TMR0 = TEN_MSEC_TIMER_TMR0;          /* Initialize timer */
      OPTION = TEN_MSEC_TIMER_OPTION;
      INTCON = GLOBAL_ENABLE;              /* Ensure interrupts are enabled */
      T0IE = 1;                                  /* Enable timer interrupt */
      while(T0IE == 1);                    /* Wait for delay to complete */
}
```

```
/*****************************************************************************
Function Name:    get_byte()
Number/Version:
History:
      Date              Rev         Author                    Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      unsigned char byte

Global Variables:
      None.

Functions Referenced:
      None.

Abstract:   Gets a byte from the KYE-13 or KOI-18.
*****************************************************************************/
unsigned char get_byte(void)
{
      /* Variable declarations */
      unsigned char bit_count, stable_count;
      unsigned char data_byte;

      /* Get one byte of fill data (clocked in on falling edge) */
      for(bit_count = 0;
            (bit_count < 8) && key_loader_present; bit_count++)
      {
            /* Wait for clock to go low */
            for(stable_count = 0;
                  (stable_count < DEGLICH_COUNT) && key_loader_present;
                  stable_count++)
                  if(fill_clk) stable_count = 0;

            /* Put data bit into data byte (MSB first) */
            if(key_loader_present)
                  data_byte = (data_byte << 1) + fill_data;

            /* Wait for clock to go high */
            for(stable_count = 0;
                  (stable_count < DEGLICH_COUNT) && key_loader_present;
                  stable_count++)
                  if(!fill_clk) stable_count = 0;
      }

      /* Return data byte */
      return data_byte;
}

/*****************************************************************************
Function Name:    check_eeprom()
Number/Version:
History:
      Date              Rev         Author                    Description
```

```
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      unsigned char *key_present_ptr

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      None.

Abstract:   Scans EEPROM for possible presence of key.
*****************************************************************************/
void check_eeprom(unsigned char *key_present_ptr)
{
      /* Variable declarations */
      unsigned char i;
      unsigned char no_data = !0;          /* Assume no data in EEPROM */
      unsigned char bad_data = 0;          /* Primary and backup data matches */
      unsigned char stored_value;

      for(i = 0; i < BACKUP_CW_STORAGE; i++)
      {
            stored_value = eeprom_read(PRIMARY_CW_STORAGE + i);
            if(stored_value && (stored_value != 0xff))
                  no_data = 0;                    /* Have data in EEPROM */
            if(stored_value != eeprom_read(BACKUP_CW_STORAGE + i))
                  bad_data = !0;                  /* Mismatch => bad key data */
      }
      if(no_data || bad_data)
      {
            *key_present_ptr = 0;    /* No good key data */

            /* Flash kgv_ok to indicate the key is no good (10 flashes/sec) */
            interrupt_function = FAST_FLASH;     /* Set up interrupt */
            TMR0 = INDICATOR_FLASH_TMR0;         /* Initialize timer */
            OPTION = INDICATOR_FLASH_OPTION;
            timer_count = FAST_FLASH_COUNT;      /* 0.05 second interval */
            T0IE = 1;
            kgv1_ok = 0;                               /* Indicator on */
#ifdef DUAL_KGV_SYSTEM
            kgv2_ok = 0;                               /* Indicator on */
#endif
      }
      else
      {
            /* Have a key */
            *key_present_ptr = !0;
      }
}

/*****************************************************************************
Function Name:    display_load_status()
Number/Version:
```

```
History:
      Date                Rev         Author                    Description
      17-Dec-1998 1.00        C. Houlberg       Baseline.

Input Variables:
      None.

Output Variables:
      None.

Global Variables:
      None.

Functions Referenced:
      None.

Abstract:   Indicates if KGV-68 was properly loaded.
*****************************************************************************/
void display_load_status(void)
{
   if(kgv1_not_loaded)
   {
         /* Flash kgv_ok to indicate the load is no good */
         interrupt_function = SLOW_FLASH;     /* Set up interrupt */
         timer_count = SLOW_FLASH_COUNT;
         TMR0 = INDICATOR_FLASH_TMR0;         /* Initialize timer */
         OPTION = INDICATOR_FLASH_OPTION;
         INTCON = GLOBAL_ENABLE;            /* Ensure interrupts are enabled */
         T0IE = 1;
         kgv1_ok = 0;                                 /* Indicator on (toggle) */
   }
   else
   {
         kgv1_ok = 0;                   /* Properly loaded (indicator on) */
   }
#ifdef DUAL_KGV_SYSTEM
      if(kgv2_not_loaded)
      {
            /* Flash kgv_ok to indicate the load is no good */
            interrupt_function = SLOW_FLASH;     /* Set up interrupt */
            timer_count = SLOW_FLASH_COUNT;
            TMR0 = INDICATOR_FLASH_TMR0;         /* Initialize timer */
            OPTION = INDICATOR_FLASH_OPTION;
            INTCON = GLOBAL_ENABLE;            /* Ensure interrupts are enabled */
            T0IE = 1;
            kgv2_ok = 0;                                 /* Indicator on (toggle) */
      }
      else
      {
            kgv2_ok = 0;                   /* Properly loader (indicator on) */
      }
#endif
}

/* end */
```

```
/*****************************************************************************
Module Name:       nvmem.h
Number/Version:    1.00
History:
      Date              Rev         Author            Description
      17-Dec-1998       1.00        C. Houlberg       Baseline.

Abstract:    Project definitions.
*****************************************************************************/

/*    Constant definitions.
*/
      /* Key Loader Data Interface Signals */
      #define sense_in         RA0          /* Signal activating KGV-68 for keying */
      #define fill_clk         RA1          /* Non-volatile memory key load clock */
      #define fill_data        RA2          /* Non-volatile memory key load data */
      #define var_req          RA3          /* Strobe requesting key load */
      #define erase            RA4          /* Analog input         2.5 Volt threshold */
      /* Key Loader Indicator Signals */
      #define kgv1_ok          RB0          /* KGV1 key load accepted and OK */
      #define erase_ind        RB1          /* Erased key indicator */
      #define kgv2_ok          RC0          /* KGV2 key load accepted and OK */
      /* System Interface Signals */
      #define flight_erase     RA5          /* Analog input         22.5 Volt threshold */
      #define xmtr_disable     RB2          /* Transmitter disable signal */
      /* KGV Interface Signals */
      #define encr_sen_in1     RB3          /* Sense signal for KGV1 */
      #define encr_fclk        RB4          /* KGV key loading clock */
      #define encr_fdata       RB5          /* KGV key loading data */
      #define encr_var_req     RB6          /* KGV key variable request strobe */
      #define encr_ran_cp1     RB7          /* KGV1 random compare OK (active low) */
      #define encr_sen_in2     RC3          /* Sense signal for KGV2 */
      #define encr_mr          RC4          /* KGV master reset */
      #define encr_ck_ok1      RC5          /* KGV1 key check OK (active low) */
      #define encr_ck_ok2      RC6          /* KGV2 key check OK (active low) */
      #define encr_ran_cp2     RC7          /* KGV2 random compare OK (active low) */

      /* Port A and B data direction (1/0 => Input/Output) */
#ifdef DUAL_KGV_SYSTEM
      #define PORT_A_DIRECTION            0x37
      #define PORT_B_DIRECTION            0xc0
      #define PORT_C_DIRECTION            0xe6
#else /* Single KGV system */
      #define PORT_A_DIRECTION            0x17
      #define PORT_B_DIRECTION            0xc0
#endif

/* end */
```

```
/*
 *    Header file for the Microchip
 *    PIC 16CR83 chip
 *    PIC 16F83 chip
 *    PIC 16C84 chip
 *    PIC 16F84 chip
 *    PIC 16CR84 chip
 *    Midrange Microcontrollers
 */

static volatile unsigned char RTCC        @ 0x01;
static volatile unsigned char TMR0        @ 0x01;
static volatile unsigned char PCL         @ 0x02;
static volatile unsigned char STATUS      @ 0x03;
static          unsigned char FSR         @ 0x04;
static volatile unsigned char PORTA       @ 0x05;
static volatile unsigned char PORTB       @ 0x06;
static volatile unsigned char EEDATA      @ 0x08;
static volatile unsigned char EEADR       @ 0x09;
static          unsigned char PCLATH      @ 0x0A;
static volatile unsigned char INTCON      @ 0x0B;

static          unsigned char bank1 OPTION      @ 0x81;
static volatile unsigned char bank1 TRISA       @ 0x85;
static volatile unsigned char bank1 TRISB       @ 0x86;
static volatile unsigned char bank1 EECON1      @ 0x88;
static volatile unsigned char bank1 EECON2      @ 0x89;

/*    STATUS bits */
static volatile bit     RP0   @ (unsigned)&STATUS*8+5;
static volatile bit     TO    @ (unsigned)&STATUS*8+4;
static volatile bit     PD    @ (unsigned)&STATUS*8+3;
static volatile bit     ZERO  @ (unsigned)&STATUS*8+2;
static volatile bit     DC    @ (unsigned)&STATUS*8+1;
static volatile bit     CARRY @ (unsigned)&STATUS*8+0;

/*      PORTA bits      */
static volatile bit      RA4     @ (unsigned)&PORTA*8+4;
static volatile bit      RA3     @ (unsigned)&PORTA*8+3;
static volatile bit      RA2     @ (unsigned)&PORTA*8+2;
static volatile bit      RA1     @ (unsigned)&PORTA*8+1;
static volatile bit      RA0     @ (unsigned)&PORTA*8+0;

/*      PORTB bits      */
static volatile bit      RB7     @ (unsigned)&PORTB*8+7;
static volatile bit      RB6     @ (unsigned)&PORTB*8+6;
static volatile bit      RB5     @ (unsigned)&PORTB*8+5;
static volatile bit      RB4     @ (unsigned)&PORTB*8+4;
static volatile bit      RB3     @ (unsigned)&PORTB*8+3;
static volatile bit      RB2     @ (unsigned)&PORTB*8+2;
static volatile bit      RB1     @ (unsigned)&PORTB*8+1;
static volatile bit      RB0     @ (unsigned)&PORTB*8+0;
static volatile bit      INT     @ (unsigned)&PORTB*8+0;

/*    INTCON bits */
static volatile bit     GIE   @ (unsigned)&INTCON*8+7;
static volatile bit     EEIE  @ (unsigned)&INTCON*8+6;
```

```
static volatile bit     T0IE  @ (unsigned)&INTCON*8+5;
static volatile bit     INTE  @ (unsigned)&INTCON*8+4;
static volatile bit     RBIE  @ (unsigned)&INTCON*8+3;
static volatile bit     T0IF  @ (unsigned)&INTCON*8+2;
static volatile bit     INTF  @ (unsigned)&INTCON*8+1;
static volatile bit     RBIF  @ (unsigned)&INTCON*8+0;

/*    OPTION bits */
static bank1 bit  RBPU        @ (unsigned)&OPTION*8+7;
static bank1 bit  INTEDG      @ (unsigned)&OPTION*8+6;
static bank1 bit  T0CS        @ (unsigned)&OPTION*8+5;
static bank1 bit  T0SE        @ (unsigned)&OPTION*8+4;
static bank1 bit  PSA         @ (unsigned)&OPTION*8+3;
static bank1 bit  PS2         @ (unsigned)&OPTION*8+2;
static bank1 bit  PS1         @ (unsigned)&OPTION*8+1;
static bank1 bit  PS0         @ (unsigned)&OPTION*8+0;

/*     TRISA bits      */
static volatile bank1 bit       TRISA4  @ (unsigned)&TRISA*8+4;
static volatile bank1 bit       TRISA3  @ (unsigned)&TRISA*8+3;
static volatile bank1 bit       TRISA2  @ (unsigned)&TRISA*8+2;
static volatile bank1 bit       TRISA1  @ (unsigned)&TRISA*8+1;
static volatile bank1 bit       TRISA0  @ (unsigned)&TRISA*8+0;

/*     TRISB bits      */
static volatile bank1 bit       TRISB7  @ (unsigned)&TRISB*8+7;
static volatile bank1 bit       TRISB6  @ (unsigned)&TRISB*8+6;
static volatile bank1 bit       TRISB5  @ (unsigned)&TRISB*8+5;
static volatile bank1 bit       TRISB4  @ (unsigned)&TRISB*8+4;
static volatile bank1 bit       TRISB3  @ (unsigned)&TRISB*8+3;
static volatile bank1 bit       TRISB2  @ (unsigned)&TRISB*8+2;
static volatile bank1 bit       TRISB1  @ (unsigned)&TRISB*8+1;
static volatile bank1 bit       TRISB0  @ (unsigned)&TRISB*8+0;

/*   EECON1 bits */
static volatile bank1 bit     EEIF  @ (unsigned)&EECON1*8+4;
static volatile bank1 bit     WRERR @ (unsigned)&EECON1*8+3;
static volatile bank1 bit     WREN  @ (unsigned)&EECON1*8+2;
static volatile bank1 bit     WR    @ (unsigned)&EECON1*8+1;
static volatile bank1 bit     RD    @ (unsigned)&EECON1*8+0;


/* macro versions of EEPROM write and read */
#define     EEPROM_WRITE(addr, value)
while(WR)continue;EEADR=(addr);EEDATA=(value);GIE=0;WREN=1;\
                            EECON2=0x55;EECON2=0xAA;WR=1;WREN=0
#define     EEPROM_READ(addr) ((EEADR=(addr)),(RD=1),EEDATA)

/* library function versions */

extern void eeprom_write(unsigned char addr, unsigned char value);
extern unsigned char eeprom_read(unsigned char addr);


#define CONFIG_ADDR     0x2007
#define FOSC0           0x01
#define FOSC1           0x02
```

```
#define WDTE            0x04
#define PWRTE           0x08

/* code protection */
#if defined (_16C84)
#define CP              0x10
#endif

#if defined (_16CR83) || defined(_16CR84)
#define DP              0x80
#define CP              0x3F70
#endif

#if defined (_16F83) || defined(_16F84)
#define CP              0x3FF0
#endif

#define UNPROTECT CP
#define PROTECT         0x0000
```